Applying Categorization to Improve Learning in Complex Environments

Alejandro Agostini[1] and Enric Celaya[2]

[1,2] Institut de Robòtica i Informàtica Industrial, Barcelona, Spain, e-mail:(agostini,celaya) @iri.upc.es

Abstract— **An autonomous agent is considered intelligent if it is capable to improve its performance through time, learning by experience. The problem of learning to improve performance from experience has been formalized in the field of Reinforcement Learning. The main difficulty found in this field is that the number of experiences required for learning to take place could be very large depending on the environment structure. If the environment has no structure at all the learning cannot be completed until all actions have been executed at least once in each possible state. Then, the application of reinforcement learning techniques is only feasible when the environment presents enough regularity to allow some kind of generalization. Many reinforcement learning techniques have been proposed using different forms of generalization. Despite the diversity of strategies proposed, usual generalization techniques do not take advantage of all the opportunities of generalization. A new algorithm was proposed that exploits a type of regularity that is denoted as categorizability. Categorizability means that from all the relevant features that must be taken into account to decide the best action in any situation, only a few of them are actually relevant in each particular situation. In this paper the categorization and learning capabilities of the algorithm are evaluated using a problem that satisfies to a good extent the categorization property. The categorization achieved is analyzed in detail and illustrated with examples. The algorithm learning performance is compared with those of other learning algorithms. Some improvements of the original algorithm are introduced.**

Index Terms— **Reinforcement Learning, Categorization, Complex Environments.**

I. INTRODUCTION

For an autonomous agent to show intelligent behaviour, it must be able to identify the relevant aspects of the environment that allow it to select the most appropriate actions according to its goals. In complex environments, the number of aspects that the agent must take into account can be very large, and usually, the agent will not have a complete knowledge about what is the best possible action in any situation. A key aspect of intelligent behaviour is the ability to improve performance through time, or learning by experience. The problem of learning to improve performance from experience has been formalized in the field of Reinforcement Learning, which constitutes an active area of research [5], [14].

The main difficulty of reinforcement learning is the number of experiences required for learning to take place. In a complex environment, the number of possible states that need to be distinguished to select the appropriate action can be very large. If we assume that the environment has no structure at all, that is, that the result of an action in a given state cannot be predicted from the experience obtained in other states, then, to learn an optimal policy, all actions must be executed at least once in each possible state. Since this is impossible in most realistic problems, the application of reinforcement learning is only feasible when the environment presents enough regularity to allow generalization and therefore reduce the amount of experiences needed.

Many reinforcement learning algorithms have been proposed using different forms of generalization. Each generalization technique implicitly assumes and exploits a possible kind of regularity that can be present in the environment, and its success depends on the degree in which the environment shows such regularity or not. Thus, for example, clustering techniques try to group a number of similar states that can be considered equivalent for the purposes of the agent, and treat them as a single state [2],[4],[6],[8]. Feature-based approaches assume that not all observable features are equally important to decide the optimal action, and build the clusters of states based on this [10],[11]. Neural net approaches assume that the mapping from states to optimal actions is sufficiently smooth and can be approximated by linear combination of the values taken in nearly states [7].

Despite the diversity of strategies proposed, reinforcement learning is still a hard task for most realistic problems. However, we observe that the usual generalization techniques used in reinforcement learning do not take advantage of all the opportunities of generalization: for example, two states are clustered together only if the result of all actions is similar for both. The existence of a single action that provides different results implies a separation in two clusters, thus preventing the generalization for all those actions that produce similar results in both states. The question, then, is: What kinds of regu-

larities can be expected in the environment and how to take advantage of them in a reinforcement learning algorithm?

In [12] a categorization and learning algorithm is proposed that exploits a type of regularity that is denoted as categorizability. In short, categorizability means that from all the features of the environment that must be taken into account to decide the best action in any situation, only a relatively small subset of them are actually relevant in each particular situation. This does not mean that some features are always irrelevant and can be ignored, but that depending on the situation, the effect of each action can be better predicted with different subsets of the whole set of features available. The assumption that the number of relevant features in each particular situation is much smaller than the total number of features that can be relevant in some situation constitutes the basis of the proposed categorization algorithm.

In this paper we introduce some improvements of the original algorithm presented in [12]. Then, we evaluate the categorization and learning capabilities of the algorithm applying it to a simple problem that satisfies to a good extent the categorization property: the game of tic-tac-toe.

The structure of the paper is the following. In section 2 we summarize the fundamental aspects of the Categorization and Learning Algorithm. Then, in section 3 some improvements of the original algorithm are presented. In Section 4 the problem formulation is made and general outlines about the implementation are given. Section 5 contains the results obtained and an evolution of the categorization capability analysis of the algorithm. Finally, the conclusions of the work are given in section 6.

II. CATEGORIZATION AND LEARNING ALGORITHM

In this section we summarize the fundamental aspects of the Categorization and Learning algorithm (CL algorithm) as presented in [12]. Some improvements we introduced in this algorithm will be explained in section 3.

It is assumed that the world is perceived through a set of n binary feature detectors f_i i=1...n. We define a *partial view of order m,* denoted by $v(f_{i1},...,f_{im})$, as a virtual feature detector that becomes *active* when its m component feature detectors are simultaneously active. The categorization process starts with the initial set of feature detectors (all partial views of order 1), and progressively builds partial views of higher order, depending on the requirements of the learning task.

For each existing partial view v, and for each action a that is executable when v is active, a value $q_v(a)$ is maintained estimating the average discounted reward obtained after executing a when v is active.

Two more values are stored for each partial view and action:

- $e_v(a)$, the estimated average absolute error of $q_v(a)$, that provides a measure of the dispersion of the actual q values obtained in the different situations in which the partial view was active. Making some simplifying assumptions, we consider that a partial view with value $q_v(a)$ and error $e_v(a)$ predicts that executing action a when v is active will result in a q value in the interval $I_v(a)=[q_v(a) - 2\ e_v(a),\ q_v(a) + 2\ e_v(a)]$.

- $i_v(a)$, the *confidence index*, that registers the number of times action a has been tried when v was active and resulted in a value of q according to the prediction. This is used to estimate a confidence value for $q_v(a)$ and $e_v(a)$ using a monotonically increasing function with saturation that takes values between 0 and $\beta < 1$, where β is a parameter of the system,

$$c_v(a) = \min\{\beta,\ confidence\_function(i_v(a))\} \qquad (1)$$

The value for which the *confidence_function*( ) reaches the saturation value β is controlled by a parameter η.

A. Action selection

As in the usual Q-learning algorithm, we must determine, for each situation, the action that maximizes the expected q value. The problem in our case is that in a given situation we may have many different predictions for the same action: one for each active partial view. To address this problem, the *relevance* $\rho_v(a)$ of partial view v for action a, is defined as

$$\rho_v(a) = \frac{1}{1+e_v(a)} \qquad (2)$$

The relevance $\rho_v(a)$ takes values in the interval (0,1) and estimates how precisely the q value for action a can be predicted by the partial view v. A perfect prediction with $e_v(a)$=0 corresponds to a relevance $\rho_v(a)$=1. Since the relevance depends on the error estimation, it is also subject to the confidence estimation. Thus, the q prediction for action a will be made according to the most relevant partial view for this action, weighted by the confidence. Therefore, we define the *winner* partial view for action a in a given situation V, as the active partial view for which the product $\rho_v(a) \cdot c_v(a)$ is maximum.

$$winner(V,a) = \arg\max_{\forall v \in V} \{\rho_v(a) \cdot c_v(a)\} \quad (3)$$

where V is the set of active partial views. In this way, the q prediction for an action in a given situation will be obtained from the winner partial view for this action.

To get an actual q prediction from the winner partial view w, two sources of uncertainty must be considered: On the first place, since each partial view predicts that the q value is expected to lay in the interval $I_w(a)$, some value in this interval is selected at random as initial guess:

$$\text{i\_guess}(a) = rand(q_w(a) - 2\, e_w(a),\ q_w(a) + 2\, e_w(a)) \quad (4)$$

On the second place, a noise term is added to account for the uncertainty of the values stored in the partial view, as evaluated by the confidence:

$$\text{guess}(a) = c_w(a)\,\text{i\_guess}(a) + (1 - c_w(a))\, rand(q_{min}, q_{max}) \quad (5)$$

where q_{min} and q_{max} are the minimum and maximum q values actually obtained so far in the learning process. Once a guess is obtained for each of the actions that are executable in the current situation, the action with highest guess is selected for execution. Note that this strategy implements an adaptive form of exploration: actions with low confidence always have some opportunity to be executed even with low q predictions, but exploratory actions have little chances to occur in the situation in which there is a strong confidence on the prediction of a high q value for some action.

B. System update

After the execution of an action, a reward r is obtained and a new situation V' is perceived, so that the actual q obtained from the execution of action a can be computed as:

$$q = r + \gamma . \max_{\forall a} \{q_v(a) \mid v = winner(V', a)\} \quad (6)$$

where γ is the discount factor. This information is used to update the estimated values for the executed action of all partial views that were active in the last situation. The $q_v(a)$ and $e_v(a)$ values are updated with identical schemas:

$$q_v(a) = c_v(a)\, q_v(a) + (1 - c_v(a))\, q \quad (7)$$

$$e_v(a) = c_v(a)\, e_v(a) + (1 - c_v(a)) \mid q - q_v(a) \mid \quad (8)$$

Note that the confidence estimation is used as a learning rate parameter, so that values with low confidence are shifted towards the observed value faster than values with higher confidence.

Finally, each confidence index $i_v(a)$ is increased by one if the actual q value lies in the predicted interval $I_v(a)$, and decreased by one in the other case.

C. Partial view generation

If the prediction of the q value is inaccurate, τ new partial views are created to help improving the prediction in the future. A prediction is considered inaccurate when the absolute difference between the predicted value $q_v(a)$ and q is higher than a user defined amount δ.

New partial views are created by combination of two already existing partial views, randomly chosen among those that were active in the last situation. This random selection is made preferring the partial views with higher confidence and with better prediction of q.

To avoid an undesired proliferation of partial views in the system, their number is limited to a threshold μ, a parameter of the system whose appropriate value depends on how much categorizable, in the sense we defined above, is the environment. To comply with this threshold, it is necessary to remove partial views when its number grows above μ. There are two different elimination criteria: *redundancy* and *creation error*.

A partial view is considered redundant when its reward predictions are too similar to the reward predictions of its parents. This redundancy is measured in the following way:

$$red(v) = \min_{\forall a} \{\max \{sim(I_v(a), I_{v_1}(a)), sim(I_v(a), I_{v_2}(a))\}\} \quad (9)$$

where v_1 and v_2 are the parent partial views and,

$$sim(I_v(a), I_{v'}(a)) = \frac{\|I_v(a) \cap I_{v'}(a)\|}{\max\{\|I_v(a)\|, \|I_{v'}(a)\|\}} \quad (10)$$

Partial views with redundancy higher than a value λ are eliminated from the system.

On the other hand, the creation error is used to eliminate partial views that are less useful for the system. This error indicates how bad the prediction was when the partial view was created. If this error is low, the created partial view is considered not so useful. Then, the partial views with lowest creation error are eliminated. The criteria used for partial view elimination may have an important impact in the performance of the algorithm, and this is the object of the improvements introduced in the next section.

III. New Partial View Elimination Criteria

A. Redundancy

The redundancy computed in equation (9) only considers the redundancy of a partial view with its parents, but in general, a partial view can be redundant with any other subset of partial views. Thus, the redundancy of a partial view is now calculated using not only its parents but also all the partial views composed by a subset of its features:

$$red(v) = \min_{\forall a}\{\max_{\forall v' \subset v}\{sim(I_v(a), I_{v'}(a))\}\} \quad (11)$$

B. Utility Index

The criterion of the creation error for the elimination of less useful partial views takes the value of the error in the prediction at the time the partial view was created. But, as the system evolves, partial views with low creation error may become more useful (or the converse) so we need a better criterion to assess the current utility of a partial view. Intuitively, the utility of a partial view could be measured taking its relevance. But, to avoid the premature elimination of partial views not sufficiently tested, we must keep partial views with low confidence. We devised a new criterion to estimate the utility of a partial view based on these statements by taking the ratio between the relevance and the confidence as a utility indicator of a partial view for an action execution.

$$u(v) = \max_{\forall a}\left\{\frac{\rho_v(a)}{c_v(a)}\right\} \quad (12)$$

IV. Problem Formulation

In order to evaluate the categorization capability of the CL algorithm we use a simple problem, the game of tic-tac-toe, that satisfies to a good extent the categorizability property. This problem is widely used to exemplify and evaluate generalization and clustering techniques [9], [1], [11], [6], and other learning methods [3], [13].

The opponent used in the training task was created with a temporal difference algorithm with state-value update [14], making an average of one exploratory move in each match and with a learning rate of 0.05. It was trained using 100.000 matches playing against itself. Then the playing policy learned is used as a fixed opponent to train the other reinforcement learning algorithms.

The reward value considered is 100 for a win, -100 for a lose, 50 for a draw, and 0 in non terminal situations.

The set of binary feature detectors consists of 3 features detectors for each cell of the board: X(i,j), O(i,j) and *empty*(i,j), where $i \in \{1,2,3\}$ indicates the number of row, and $j \in \{1,2,3\}$ the number of column. Each of these features becomes active when the corresponding cell contains the value X, O or is empty, respectively. Actions are also represented as an X or an O in a particular position (i,j).

To exemplify the categorizability of the tic-tac-toe game note that in the states shown in Fig. 1 only features X(1,1) and X(2,2) are relevant to determine that the match is won by playing an X in position (3,3).

X	O	O
O	X	
X		

X		
	X	O
O		

Fig 1. Two different situations in the game of tic-tac-toe categorizable with the same partial view.

A point to remark is that the action selection is performed only over available actions. Thus, if one cell is occupied with an X or O then it is not considered as a possible place to move. Finally, the CL algorithm is trained playing with X's and always plays in the first place.

After some empirical experiments, the following set of training parameters for the CL algorithm was selected: β=0.99, η=50, δ= 50, λ=0.90, μ=500, γ=0.90 and τ=3.

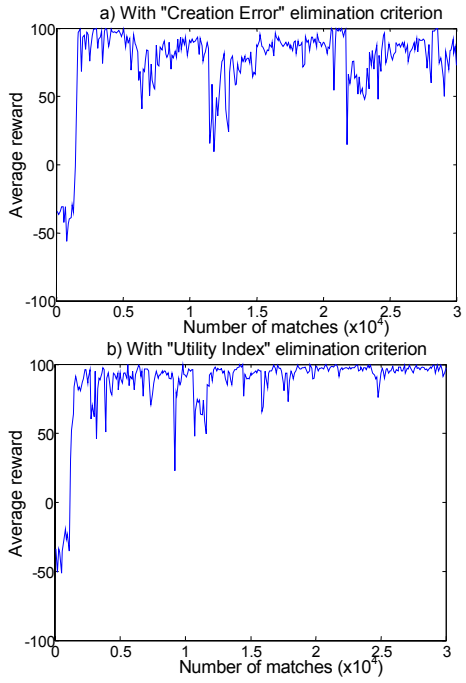

Fig 2. Average reward obtained during the learning task: a) with "Creation error" elimination criterion, and b) with "Utility index" elimination criterion.

V. RESULTS

Fig. 2 shows the average rewards obtained with the CL algorithm along 30.000 training matches before and after the modifications explained in section 3. As we can observe in Fig. 2 a), the average reward trend shows some instability in the learning process. After performing the mentioned changes a new training process was done obtaining the results shown in Fig. 2 b). As observed, the learning process is much more stable.

A. Comparison with Q-Learning

Fig. 3 shows the average reward trends over 30.000 matches of CL algorithm and Q-Learning algorithm [15]. For Q-Learning, a learning rate of 0.05, a discounted coefficient of 0.9 and an average of one exploratory move per match were used.

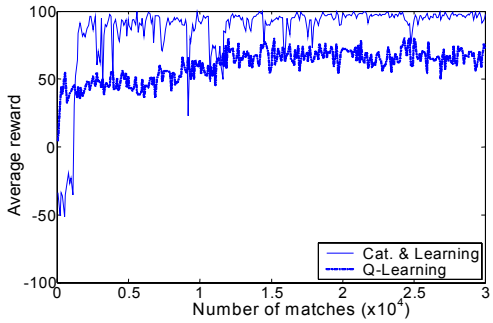

Fig 3. Performance comparison between Q-Learning and CL algorithms.

We observe that, at the beginning of the training, the CL algorithm fails to categorize the tic-tac-toe environment using the initial partial views, composed of only one feature. Then, new partial views were generated improving categorization and consequently increasing the average reward. After the convergence is reached, the CL algorithm performs better than Q-Learning. Nevertheless, differences in the exploration methods influence this final average reward value.

In order to obtain a better comparison of the algorithm performance, 100 matches are played against the training opponent using both learned sets. No exploration is made except the first move that is made in a random position. Table 1 shows the obtained results.

TABLE 1. RESULTS OF 100 MATCHES PLAYED AGAINST THE TRAINING OPPONENT

	Cat. & Learning	**Q-Learning**
Won	100	85
Drawn	0	15
Lost	0	0

CL algorithm won all the matches while Q-Learning drew 15 of them. With this opponent, CL algorithm learned a better game policy than Q-Learning.

It is remarkable that Q-Learning experimented and stored about 6.000 states against the 500 partial views stored by the CL algorithm.

B. Partial Views Generated

Fig. 4 shows some partial views generated with the CL algorithm. Each board only shows the features of the partial view marked with gray and the corresponding qv, ev and iv values associated to each possible action.

a)

-	-	-
-	**X**	-
X	-	-

b)

-	**O**	-
-	**O**	-
-	-	-

a	$q_v(a)$	$e_v(a)$	$i_v(a)$
(1,1)	*75,7*	*41,9*	*50*
(1,2)	*61,2*	*25,5*	*50*
(1,3)	*100,0*	*0,0*	*50*
(2,1)	*78,3*	*24,3*	*49*
(2,2)	-	-	-
(2,3)	*24,7*	*69,9*	*50*
(3,1)	-	-	-
(3,2)	*11,9*	*50,6*	*50*
(3,3)	*71,6*	*15,5*	*49*

a	$q_v(a)$	$e_v(a)$	$i_v(a)$
(1,1)	*-100*	*0,0*	*21*
(1,2)	-	-	-
(1,3)	*-100*	*0,0*	*16*
(2,1)	*-100*	*0,0*	*48*
(2,2)	-	-	-
(2,3)	*-100*	*0,0*	*25*
(3,1)	*-100*	*0,0*	*50*
(3,2)	*56,6*	*22,7*	*50*
(3,3)	*-100*	*0,0*	*30*

Fig 4. Generated partial views of order 2 and their associated action values.

It is clear that these generated partial views contain only relevant features. Note that in Fig. 3 a) the action that win the match is that with highest qv(a) (in fact the maximum possible), lowest ev(a) (highest relevance) and high confidence. Fig. 4 c) shows that the algorithm also learns to avoid a lose as we can see in the values associated to action X(3,2).

The learned set contains 90 partial views of order 2, 131 of order 3, 138 of order 4, 79 of order 5, 23 of order 6, 9 of order 7, and 3 of order 8, in addition to the 27 initial partial views of order 1. It is remarkable that the mean number of features per partial view is 3.5, against the 9 features present in each situation.

C. Evaluation of the Categorization Capability

In order to evaluate the categorization capability of the CL algorithm, two complete matches against the training opponent are presented in Fig. 5. In these tests no exploration is made except for the initial moves, which are made at random positions. The CL algorithm plays with X's and each board represents a state of the game. When it is the turn of the CL algorithm the winner partial view for the selected action is marked in light gray.

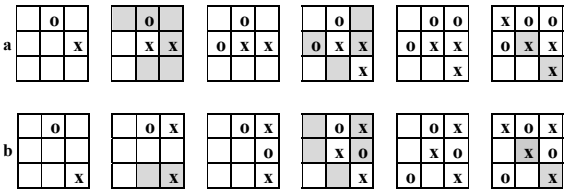

Fig 5. Two tic-tac-toe matches against the training opponent using the learned set of partial views.

We see that the winner partial views at terminal situations clearly contain relevant features, and the actions selected are those that win the match. In early states of the game there are other partial views whose relevance is not obvious to select the action. Nevertheless, they contain the relevant features to play with this particular opponent and take advantage of its imperfections.

Note that in both terminal situations the same partial view is used to select the action demonstrating the categorization achieved by the algorithm.

VI. CONCLUSION

The CL algorithm was created with the aim of taking advantage from the categorizability properties of a complex environment to improve the learning task.

From Fig. 3 and Table 1 we observed that the CL algorithm is capable to learn an adequate policy for the selected problem with a performance superior to the Q-learning algorithm. Note from table 1 that the learned policy approaches better the optimal one, always winning the game. Additionally, The CL algorithm learned to play using only 500 partial views, against the 6.000 states needed in Q-Learning for the same learning task. It is also remarkable that the mean order of these partial views is 3.5 features against the 9 features present in each situation. These last two facts imply a great improvement in the computing performance reducing the amount of memory needed to store and process the learning data.

From these points we can conclude that the CL algorithm is capable to take advantage of the categorizatibility of an environment to improve the learning task.

Additionally, despite the simplicity of the selected problem, the results obtained suggest that the CL algorithm could be suitable to be applied in a more complex categorizable environment. As a future work we would like to apply this algorithm in such a complex environment, more precisely to face the problem of locomotion control of legged robots.

REFERENCES

[1] D. Aha. Incremental Constructive Induction: An Instance-Based Approach. Machine Learning, pp 117-121, 1991.

[2] J. Boyan, A. Moore. Generalization in Reinforcement Learning: Safely Approximating the Value Function. Advances in Neural Information Processing Systems 7. MIT Press, 1995.

[3] J. Boyan. Modular Neural Network for Learning Context-Dependent Game Strategies. Master of Philosophy thesis. Department of Engineering and Computer Laboratory. University of Chicago, 1992.

[4] D. Chapman, L. Kaebling. Input Generalization in Delayed Reinforcement Learning: An Algorithm and Performance Comparisons. In Proceeding of the International Joint Conference on Artificial Intelligence, 1991.

[5] L. Kaebling, M. Littman, A. Moore. An Introduction to Reinforcement Learning. Journal of Artificial Intelligence Research, 4:237-285, 1996.

[6] M. Ginsberg. Partition Search. In Proceeding of AAAI, 1: 228-233, 1996.

[7] L. Lin. Hierarchical Learning of Robots Skill by Reinforcement. In Proceeding of the International Conference of Neural Networks, 1993.

[8] S. Mahadevan, J. Connel. Automatic Programming of Behaviour-Based Robots Using Reinforcement Learning. Artificial Intelligence, 55:311-363, 1992.

[9] C. Matheus, L. Rendell. Constructive Induction On Decision Trees. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pp 645-650, 1989.

[10] A. McCallum. Reinforcement Learning with Selective Perception and Hidden State. PhD thesis, Department of Computer Science, 1995.

[11] Y. Miyamoto, K. Uehara. Discovering New Features to Improve Q-Learning More Efficiently. CS24 Technical Report, Department of Computer and Systems Engineering, Kobe University, 1999.

[12] J. Porta, E. Celaya. Learning in Categorizable Environments. Sixth International Conference on Simulation of Adaptive Behaviour: From Animals to Animats, Paris, September, 2000.

[13] S. Siegel. Training an artificial neural network to play tic-tac-toe. ECE 539 Term Project. Computer-Aided Engineering Center. University of Wisconsin-Madison, College of Engineering, USA, 2001.

[14] R. Sutton, A. Barto. Reinforcement Learning: An Introduction, "A Bradford Book", MIT Press, 1998.

[15] C. Watkins, P. Dayan. Q-Learning. Machine Learning, 8:279-292, 1992.